Learning Regular Expressions for the Extraction of Product Attributes from E-commerce Microdata

Petar Petrovski, Volha Bryl and Christian Bizer

University of Mannheim, Germany
Research Group Data and Web Science
{petar,volha,chris}@informatik.uni-mannheim.de

Abstract. A large number of e-commerce websites have started to markup their products using standards such as Microdata, Microformats, and RDFa. However, the markup is mostly not as fine-grained as desirable for applications and mostly consists of free text properties. This paper discusses the challenges that arise in the task of matching descriptions of electronic products from several thousand e-shops that offer Microdata markup. Specifically, our goal is to extract product attributes from product offers, by means of regular expressions, in order to build well structured product specifications. For this purpose we present a technique for learning regular expressions. We evaluate our attribute extraction approach using 1.9 million product offers from 9,240 e-shops which we extracted from the Common Crawl 2012, a large public Web corpus. Our results show that with our approach we are able to reach a similar matching quality as with manually defined regular expressions.

Keywords: Feature Extraction, Entity Linking, Microdata

1 Introduction

Recently more and more websites have started to embed structured data describing various items into their HTML pages using markup standards, such as Microdata[1] [6], Microformats[2] and RDFa [1]. This results in millions of records from thousands of data sources becoming publicly available on the Web. Being able to integrate the data, e.g. in the e-commerce domain, would enable the creation of powerful applications.

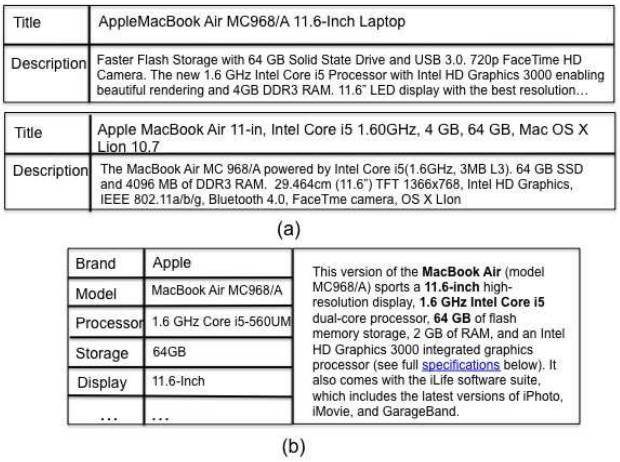
In this paper our use case is the Common Crawl corpus[3], the largest and most up-to-date publicly available web corpus, which, among others, contains product data from 9,000 e-shops. Bizer et al. [4] have successfully extracted the structured data from the 2012 Common Crawl into the WebDataCommons (WDC)

[1] Microdata is a standardized HTML extension for marking up the structured data within web pages so that it can be easily parsed by computer programs.

[2] http://microformats.org/

[3] Common Crawl – http://commoncrawl.org/

data set[4], finding that almost 15% percent of the pages contain structured data. Among the top topical domains of the data is the e-commerce domain with more than 9,000 e-shops using structured data. Bringing the information from these disparate data sets into a common integrated dataset, e.g. an open e-commerce product catalog, would substantially increase the value of the information collected.



Title	AppleMacBook Air MC968/A 11.6-Inch Laptop
Description	Faster Flash Storage with 64 GB Solid State Drive and USB 3.0. 720p FaceTime HD Camera. The new 1.6 GHz Intel Core i5 Processor with Intel HD Graphics 3000 enabling beautiful rendering and 4GB DDR3 RAM. 11.6" LED display with the best resolution...

Title	Apple MacBook Air 11-in, Intel Core i5 1.60GHz, 4 GB, 64 GB, Mac OS X Lion 10.7
Description	The MacBook Air MC 968/A powered by Intel Core i5(1.6GHz, 3MB L3). 64 GB SSD and 4096 MB of DDR3 RAM. 29.464cm (11.6") TFT 1366x768, Intel HD Graphics, IEEE 802.11a/b/g, Bluetooth 4.0, FaceTme camera, OS X LIon

(a)

Brand	Apple	This version of the **MacBook Air** (model MC968/A) sports a **11.6-inch** high-resolution display, **1.6 GHz Intel Core i5** dual-core processor, **64 GB** of flash memory storage, 2 GB of RAM, and an Intel HD Graphics 3000 integrated graphics processor (see full specifications below). It also comes with the iLife software suite, which includes the latest versions of iPhoto, iMovie, and GarageBand.
Model	MacBook Air MC968/A	
Processor	1.6 GHz Core i5-560UM	
Storage	64GB	
Display	11.6-Inch	
...	...	

(b)

Fig. 1: (a) WDC product offers; (b) Amazon product specification with the additional textual description

Figure 1a shows two product offers from the WDC dataset coming from two different e-shops describing the same product. Matching these product offers is a non-trivial task [9–11, 16] since (1) product descriptions often follow different patterns and/or different levels of detail (the top product description contains less technical description than the bottom one); (2) numeric values are often imprecise, e.g. due to rounding (the top product description contains an 11.6 inch laptop versus the 11 inch laptop in the bottom one); (3) abbreviations are used differently (the bottom product description uses SSD as abbreviation, while the top refers to the full name Solid State Drive).

In this paper we present a feature extraction method, in order to get more fine-grained structured data as an input for entity linking tools such as e.g. Silk [7], and thus improve the matching precision.

In [16] we have presented an approach covering the process of integration of product data, where we proposed feature extraction methods as a key preprocessing step for entity linking (matching).

[4] http://www.webdatacommons.org/

The feature extraction method we proposed in [16] either require manual configuration, and thus, good understanding of the input data, or are not able to extract values that are not present in the training data. Therefore, in this paper we propose a method that relies on learning regular expressions to extract product attributes from product offers, in order to build well structured product specifications for product matching. We explore a genetic programming approach for learning regular expressions.

The rest of this paper is structured as follows: In Section 2 we discuss the state of the art in the area of product matching and learning regular expressions for feature extraction. Section 3 gives the problem description and introduces the automated technique for learning regular expressions. In Section 4 we report on the evaluation of the proposed approach measuring the accuracy of the extracted attributes as well as the product matching performance. Finally, Section 5 concludes the paper.

2 Related Work

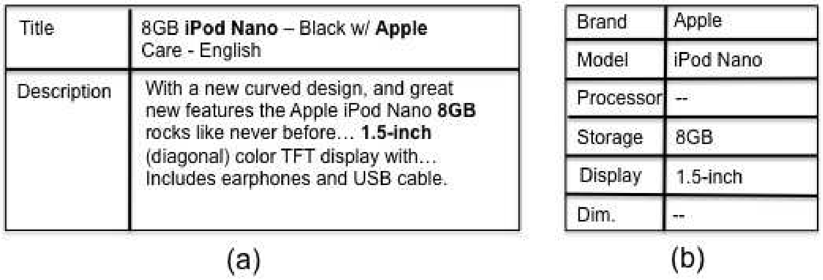
The problem of feature extraction has been studied extensively under the topic of entity disambiguation including product matching [9,11,16]. Specifically, Köpcke et al. [11] perform property mapping on product offers. While the domain is the same as ours, only free-text properties were used for entity resolution in the study. Product features were extracted from the title by manually defining regular expressions. Similarly, record linkage between free-text product offers and structured product specifications has been studied in [9]. Structured product specifications were used to learn lists of property values as a model for extracting new products features from the product offers and labeling. Even though the approach shows promising results, it lacks the ability to extract feature values which have not been present in the training set. Petrovski et al. [16] propose a combination of the previous two, by allowing different product properties to be extracted by different extraction methods.

Differently from [9, 11, 16] in this paper we propose an approach that follows automatic induction of deterministic regular expressions [2, 14], i.e. using genetic programming (GP) to learn regular expressions from examples in order to perform feature extraction. The problem of inducing regular expressions from positive and negative examples has been studied in the past, even outside the context of feature extraction [3, 13, 17]. Most of the studies assume very strong pattern in the examples, and thus the problem reduces to learning simple regular expressions. For instance, applications motivated by DNA and RNA [13] view input sequences as multiple atomic events, where each atomic event is a simple regular expression. In a similar manner, in DTD inference [3] documents are described using simple DTDs, thus again simple regular expressions are often enough to capture a DTD definition. However, regular expressions for information extraction rely on more complex constructs. Li et al. [14] introduces a novel evolutionary approach to learn regular expressions for information extraction, starting from handful of seeds. The study presents experiments mainly on properties with strong patterns like telephone numbers, e-mails and software names. Similarly, to our approach Bartoli et al. [2] use GP to learn regular expressions from examples. However it differs from our approach in that it is using different fitness function and does not mention any specific breeding techniques which are commonly known to boost performance of GP algorithms.

3 Methodology

3.1 Problem Description

We have a set D of product offers, represented as RDF statements. Every product offer $d \in D$ consists of set of properties (property-name, value). The properties most frequently found are: title, with 86% usage, and description with 64% usage. Both these properties often have an unstructured free text as values. An example of such product description can be seen in Figure 2a. D represents a subset of the WDC dataset containing more than 1.9 million product descriptions originating from 9,240 e-shops.



Title	8GB **iPod Nano** – Black w/ **Apple** Care - English
Description	With a new curved design, and great new features the Apple iPod Nano **8GB** rocks like never before… **1.5-inch** (diagonal) color TFT display with… Includes earphones and USB cable.

(a)

Brand	Apple
Model	iPod Nano
Processor	--
Storage	8GB
Display	1.5-inch
Dim.	--

(b)

Fig. 2: Example of an extracted offer: (a) WDC product offer; (b) Extracted properties

On the other hand, we have a set S of product specifications from the Amazon product catalog, where every specification consists of well defined properties. The properties of a product specification can be of numerical or categorical nature, as can be seen in the product specification example in Figure 1b. In addition, the specifications contain a textual description similar to the one found in D. Our objective is to extract new product specifications from d, shown in Figure 2b, in order to match them against the already existing specifications in S.

As in [11], our key observation is that product descriptions frequently contain product properties such as: product brand, product model, height, weight etc. Since most of these properties follow a pattern we approach the extraction problem by learning regular expressions for specific properties from s and applying the same regular expressions to the free text in d.

3.2 Learning Regular Expressions from Examples

Similarly to [2], we represent each valid regular expression as a tree by defining, for each operator of a regular expression, sub-trees suitable for that operator. The function set consists of the following regular expressions operators:

– concatenate node - a binary node that concatenates other nodes or leaves;
– possessive quantifiers - quantifier is possessive by placing an extra + after it, making the quantifier greedy;
  - ”*+” - a greedy zero or more repetitions of the preceding element;
  - ”++” - a greedy one or more repetitions of the preceding element;
  - ”?+” - a greedy zero or one repetitions of the preceding element;
  - ”{m,n}+” - a greedy matching of the preceding element at least m and not more than n times;
– the group operator ”()”;
– the character class ”[]”.

The terminal set used for the leaves of the tree consists of:

– constants - a single character, a number or a string,
– ranges - ”a-z”, ”A-Z”, ”0-9” or ”a-z0-9”,
– character classes - ”\w” or ”\d”,
– white spaces -”\s”,
– the wildcard - the ”.” character.

As input the algorithm takes a set of examples. Each example is composed of a pair of strings: *(text, the string we want to extract)*. For instance in Figure 1b the pair for the Display property would be (”[the whole textual description]”, ”11.6-inch”). An example is considered negative in the case the string we want to extract is empty. As an initial population the algorithm takes 2 times the size of the training set, or $2 * |T|$. Half of the population is generated from the examples themselves, by changing every character sequence by \w and each number sequence by \d. The other half of the initial population is generated randomly by the ramped half-and-half method [12]. This method uses two methods to create trees: (a) the full method producing full/bushy trees and (b) the grow method producing diverse tree structures with some branches longer than others. The maximum depth of the trees generated is ramped, so that individuals are created in a range of sizes. Using this method allows creating a diverse initial population in terms of structure.

The quality of a learned regular expression is assessed by the fitness function based on user-provided training data. The prediction of the regular expression is compared with the positive examples while counting true positives (TP) and false negatives (FN), and the negative examples while counting false positives (FP) and true negatives (TN). Based on these counts, a fitness value between -1 and 1 is assigned to an individual regular expression by calculating Matthews correlation coefficient (MCC):

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

In contrast to many other popular fitness measures such as the F-measure (i.e. the harmonic mean of precision and recall), Matthews correlation coefficient yields good results even for heavily unbalanced training data.

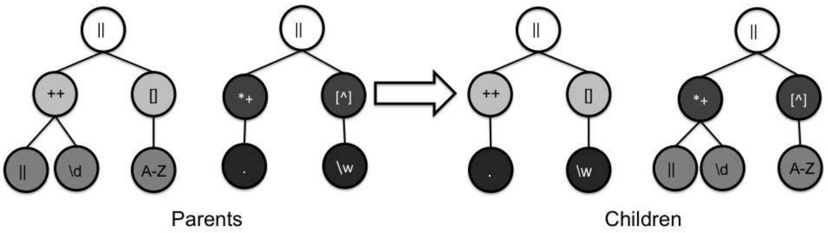


Fig. 3: Example of two point crossover

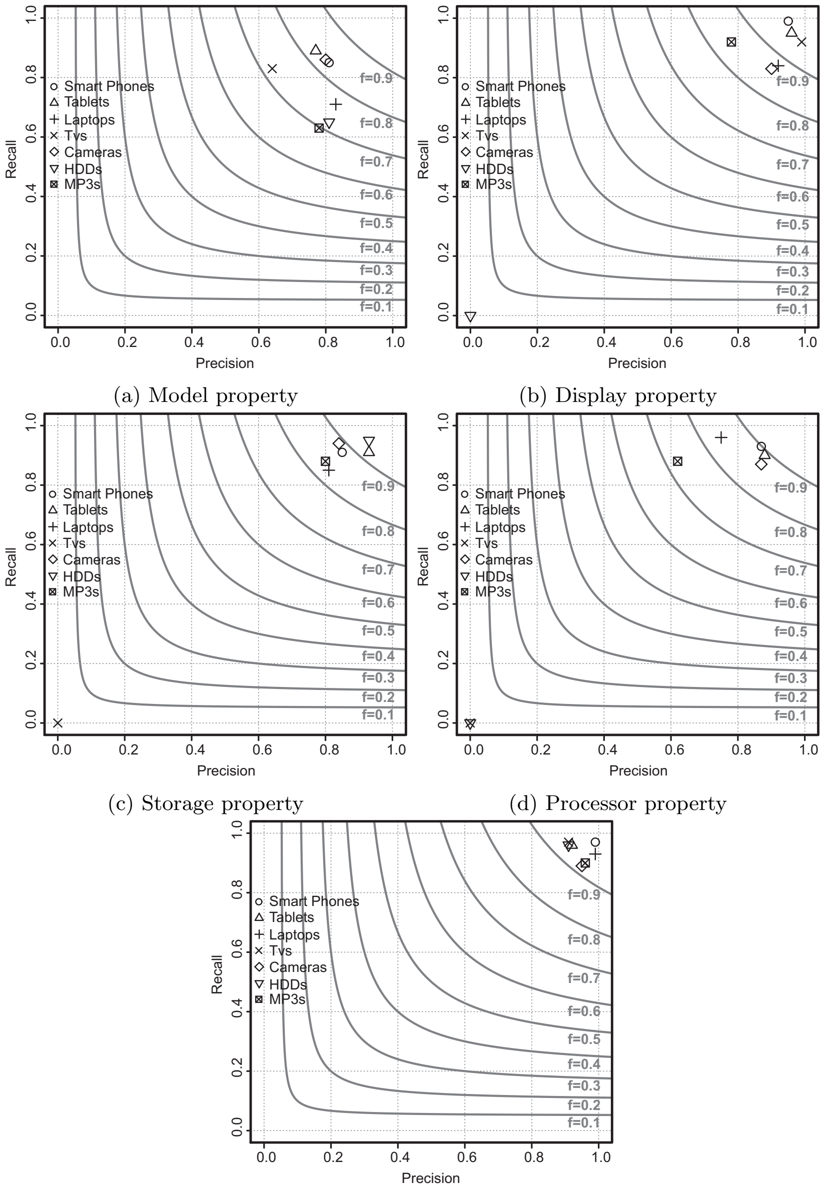
To improve the population, our approach makes use of two of the most common genetic operations: crossover and mutation. A crossover operator is used to learn more complex trees by selecting a random path of nodes in two individuals. It then combines both paths by executing a two point crossover. A two point crossover, shown in Figure 3, is executed by selecting two nodes on the parents and swapping the sub-trees between these nodes, rendering two children. Selection of the individuals is done by the tournament selection method [15], which involves splitting the population into groups and running several tournaments among the individuals in the groups. The winners from the tournaments are selected for crossover. The mutation operator is implemented similarly by selecting the crossover operator and executing a headless chicken crossover [8] i.e. crossing an individual from the population with a randomly generated one.

4 Evaluation

In this section, we present performance results from two experiments using the approach presented in the previous section. The first experiment, presented in Section 4.1, involves the use of our approach to extract specific properties from a set of product offers. In addition the second experiment, presented in Section 4.2, involves product matching where we use the output from the first experiment and match it to a subset of the already existing product specifications. The data used in the experiments and the implementation of our approach can be found at `http://www.webdatacommons.org/structureddata/2012-08/data/product/howto.html`.

4.1 Property extraction

We use a set of 5,000 electronics product offers from the WDC product dataset; the same dataset as in [16]. The training set T consists of 500 product specifications, as shown in Figure 1b, from the Amazon product catalog. The electronics



(a) Model property

(b) Display property

(c) Storage property

(d) Processor property

(e) Dimension property

Fig. 4: F-measure plots per extracted property

product offers are selected to closely match the Amazon product catalog. This was done by performing a simple pair wise matching (see the baseline method in Section 4.2) and manually annotating offers that are rightly matched. The Amazon product specifications are selected from the first 500 featured electronics products on their website. The effort to create the input examples consisted of pairing the textual description of the product specification with the property value for the property that is being learned. The experiment involved learning regular expressions for 5 different properties from T: Model, Display size (in inches), Processor, Storage size, and Dimension. The algorithm was set to run for not more than 100 iterations and stop if the best fitness is reached ($MCC = 1$). Subsequently the learned regular expressions are applied to the whole text (title and description) of the product offers.

In the following we list the learned regular expressions:

- Model - `(?:[^\d]+\s[a-z0-9]+)*+`
- Storage - `(?:\d+[^B]+[B]+)++`
- Display - `\d+.[^nc]*+nc[^o]*+`
- Processor - `\d+\s?[^z]++z`
- Dimension - `\d[^.]x?[\d]++`

Figure 4 shows the F-measure for each of the 7 of the most popular product categories (Smart Phones, Tablets, Laptops, TVs, Digital Cameras, HDDs, MP3s) and for each property. This experiment indicates that this approach yields good results, when the properties are of numeric or semi-numeric nature. Generally, if the property is numeric or a simple combination of numbers and letters, our approach performs with 89.4% F-measure (as can be seen from Figures 4b to 4e) on average. The Dimension property has the highest 94.2% F-measure, which is expected since dimension follows a strong pattern in "length x height x thickness". On the other hand, in Figure 4e, the F-measure of the Model property, which does not show a strong pattern (model can be only character based, e.g. iPpod Nano), has an F-measure of 77.2% on average for all product categories.

4.2 Product matching

The second experiment showcases the application of our property extraction approach as a preprocessing step for product matching. We use the same set of 5,000 electronics product offers from the WDC dataset as in the previous section, and we match them against 20 products from the Amazon product catalog (see Figure 1b for an example). We use the Silk link discovery framework[5] for generating linkage rules and matching the products [7]. As a baseline for the matching task we execute pairwise matching using just the title and description from both WDC and Amazon datasets with Jacard similarity as our similarity measure. In order to get a more precise comparison we extract patterns from the

[5] Silk is an open source tool for generation and learning linkage rules – http://wifo5-03.informatik.uni-mannheim.de/bizer/silk/

title and description. From the title we extract the product brand and model by matching a regular expression: `^.*(\w+_[a_zA-Z0-9]+)_\d.*(gb|hd|p[x]|inche?s?|m).*\$`. From the description we find a number/unit of measurement pairs, which usually correspond to numeric attributes like 5 m, 3.5 inches, 256 MB, etc.

The second setting involves applying the learned regular expressions on the 5,000 electronics product offers in order to get new product specifications each containing at most 5 properties (the output from the previous experiment). In most cases the regular expression matches to one or none values, however in the case of multiple matches we use a simple approach of choosing the first match.

Table 1: Precision, Recall, and F-Measure per Configuration

Configuration	Precision %	Recall %	F-measure %
Baseline	69	90	78.1
Learned Regular Expr.	80	84	81.9

Table 1 shows the precision, recall and F-measure for the two configurations. As can be seen, there is a big improvement in precision and F-measure when using our automated technique for feature extraction. The precision and F-measure are comparable to the numbers we report in [16] for the case of feature extraction with manually created regular expressions: 82% precision and 80.9% F-measure. Therefore, we can conclude that our approach reaches a similar matching quality, without the need of manually assigning regular expressions, which requires knowledge about the data as well as regular expression syntax.

5 Conclusion

This paper presents an approach for extracting product features by learning regular expressions. The evaluation indicates that this approach yields good results when the properties are of numeric or semi-numeric nature, even though the approach also proved competent when learning a regular expressions for more complex properties. Moreover, the study shows that learning regular expressions for feature extraction reaches a similar matching quality compared to the case of feature extraction with manually created regular expressions, which requires knowledge about the data and regular expression syntax.

There are a number of potential future research directions. We currently do a selection of the first match in case there are several matches when it comes to the extraction. Ideally, we would like to rank all matches in order to improve the extraction. One possibility of attaining this would be to perform semantic parsing [9], where each match is compared to tagged values in a knowledge base. Another direction is studying the effect of the elitist strategy [5], i.e. keeping the top 1% of the population in the next iteration when it comes to breeding. Finally, it would be interesting to apply the proposed approach to other topical domains, such as local businesses, postal addresses, etc.

References

1. Ben Adida and Mark Birbeck. RDFa primer - bridging the human and data webs - W3C recommendation. http://www.w3.org/TR/xhtml-rdfa-primer/, 2008.
2. Alberto Bartoli, Giorgio Davanzo, Andrea De Lorenzo, Marco Mauri, Eric Medvet, and Enrico Sorio. Automatic generation of regular expressions from examples with genetic programming. In *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '12, pages 1477–1478, 2012.
3. Geert Jan Bex, Wouter Gelade, Frank Neven, and Stijn Vansummeren. Learning deterministic regular expressions for the inference of schemas from XML data. *ACM Trans. Web*, 4(4):14:1–14:32, September 2010.
4. C. Bizer, K. Eckert, R. Meusel, H. Mühleisen, M. Schuhmacher, and J. Völker. Deployment of RDFa Microdata and Microformats on the Web – A Quantitative Analysis. In *12th International Semantic Web Conference In-Use track*, pages 17–32, 2013.
5. Moises G. de Carvalho, Alberto H. F. Laender, Marcos Andre Goncalves, and Altigran Soares da Silva. A genetic programming approach to record deduplication. *IEEE Trans. Knowl. Data Eng.*, 24(3):399–412, 2012.
6. I. Hickson. HTML Microdata. http://www.w3.org/TR/microdata/, 2011.
7. R. Isele and C. Bizer. Learning Linkage Rules using Genetic Programming. In *6th International Workshop on Ontology Matching*, pages 1638–1649, 2011.
8. Terry Jones. Crossover, macromutation, and population-based search. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 73–80. Morgan Kaufmann, 1995.
9. A. Kannan, I. Givoni, R. Agrawal, and A. Fuxman. Matching unstructured offers to structured product descriptions. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 404–412, 2011.
10. H. Köpcke, A. Thor, and E. Rahm. Evaluation of entity resolution approaches on real-world match problems. In *Proc. 36th Intl. Conference on VLDB / Proceedings of the VLDB Endowment 3(1)*, pages 484–493, 2010.
11. H. Köpcke, A. Thor, and E. Rahm. Tailoring entity resolution for matching product offers. In *Proc. 15th Intl. Conference on Extending Database Technology (EDBT)*, pages 545–550, 2012.
12. John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, USA, 1992.
13. W. B. Langdon, J. Rowsell, and A. P. Harrison. Creating regular expressions as mRNA motifs with GP to predict human exon splitting, 2009.
14. Yunyao Li, Rajasekar Krishnamurthy, Sriram Raghavan, Shivakumar Vaithyanathan, and H. V. Jagadish. Regular expression learning for information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '08, pages 21–30, Stroudsburg, PA, USA, 2008. Association for Computational Linguistics.
15. Brad L. Miller, Brad L. Miller, David E. Goldberg, and David E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9:193–212, 1995.
16. Petar Petrovski, Volha Bryl, and Christian Bizer. Integrating product data from websites offering Microdata markup. In *Proceedings of the 4th Workshop on Data Extraction and Object Search (DEOS), WWW 2014*, pages 1299–1304, 2014.
17. Borge Svingen. Learning regular languages using genetic programming. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 374–376. Morgan Kaufmann, 1998.